Game Timing and Multicore Processors

Chuck Walbourn
SDE, Windows Gaming & Graphics
Microsoft Corp

November 2005

Background

Since the introduction of the x86 P5 instruction set, many game developers have been making use of the `RDTSC` instruction to perform high-resolution timing. The Windows multimedia timers are precise enough for sound and video processing, but with frame times of a dozen milliseconds or less they don't have enough resolution to provide delta-time information. Many games still use a multimedia timer at start-up to establish the frequency of the CPU, and use that value to scale results of `RDTSC` to get accurate time. Due to the limitations of `RDTSC`, the Windows API exposes the more correct way to access this functionality through the `QueryPerformanceCounter` and `QueryPerformanceFrequency` routines.

This usage of `RDTSC`-style timing suffers from three fundamental issues:

1. `Using RDTSC` directly assumes the thread is always running on the same processor. Multiprocessor and dual-core systems do not guarantee synchronization of their cycle counters between cores. This is exacerbated when combined with modern power management technologies idling and restoring various cores at different times resulting in the cores typically being out of sync. For an application this generally results in glitches or potentially crashes as the thread jumps between the processors and gets timing values that either result in large deltas, negative deltas, or halted timing.

2. `RDTSC` locks the timing information the application requests to the processor's cycle counter. For many years this was the best way to get high-precision timing information, but newer motherboards are now including dedicated timing devices which provide high resolution timing information without the drawbacks of `RDTSC`.

3. The assumption is often made that the frequency of the CPU is fixed for the life of the program. With modern power management technologies, this is an incorrect assumption. While initially limited to laptop/mobility parts, this technology is being aggressively used in many high-end desktop PCs and it being disabled is generally not acceptable by users.

Copyright © 2005 by Microsoft Corp

Recommendations

Games need accurate timing information, but also need to implement their timing code in a way that avoids the problems associated with `RDTSC` usage. The following steps should be taken when implementing use of high-resolution timing:

1. Use the `QueryPerformanceCounter` and `QueryPerformanceFrequency` API instead of the `RDTSC` instruction. These APIs may make use of `RDTSC`, but might instead make use of a motherboard timing chip or some other system services that will provide quality high-resolution timing information.

2. When computing deltas, the values should be clamped to ensure any bugs in the timing values do not cause crashes or unstable time-related computations. The clamp range should be from 0 (to prevent negative delta values) to some reasonable value based on your lowest expected framerate. Note: This clamping is likely to be useful in any case when doing debugging of your application, but be sure to keep it in mind if doing performance analysis or running the game in some unoptimized mode.

3. Finally while the `QueryPerformanceCounter` / `QueryPerformanceFrequency` API is intended to be multiprocessor aware, bugs in the BIOS or motherboard drivers may result in these routines returning different values as the thread moves from one processor to another. We recommend that all game timing be computed on a single thread, and that thread is set to stay running on a single processor through the `SetThreadAffinityMask` Windows API. Typically this would be the main game thread. All other threads should be designed to operate without gathering their own timer data. We do not recommend using a 'worker' thread to compute timing as this will become a synchronization bottleneck. We also do not recommend have multiple threads compute timing and associating each with a specific processor, as this will greatly reduce the throughput on multicore systems.

Application Compatibility Solutions

Many game developers have made assumptions about the behavior of `RDTSC` over many years, so it is quite likely that some existing applications will exhibit problems when run on a multiprocessor/multicore system due to their timing implementation. This will usually express itself as glitching or slow-motion movement. There is no easy remedy for a lack of being power management aware, but there is an existing shim for forcing an application to always run on a single processor even on a multiprocessor system.

To create this shim, download the *Microsoft Application Compatibility Toolkit* from http://www.microsoft.com/windows/appcompatibility/default.mspx.
Using the Compatibility Administrator, you create a database of your applications and associated fixes, and then save out that database file. To force all the threads of the

Copyright © 2005 by Microsoft Corp

application to run on a single processor/core use the `SingleProcAffinity` fix. Using the command-line tool `fixpack.exe`, you can convert this database into an installable package for installation, test, and distribution. See the toolkit's documentation for more detail.

Copyright © 2005 by Microsoft Corp